Game Theory and The Prisoner's Dilemma

Alec Yu
Mark Chae

Introduction

Game theory plays a large role in the analysis of human behaviors. It has various applications in fields like economics, politics, biology, computer science, philosophy, and many more. Game theory formally began with a paper written by John von Neumann, a Hungarian-American mathematician. This paper then became a centerpiece for game theory and mathematical economics. The scope of game theory was then widened with the emergence of the Cold War. In 1950, the RAND Corporation ran mathematical discussions and experiments on the prisoner's dilemma, a game problem that we will be further exploring in this write-up, in order to develop applications into nuclear strategy. It was soon after that John Nash developed a revolutionary criterion for player strategies called the Nash Equilibrium. We will explore how best to decide on strategies and actions to take for a player in some two-player games. First, we will walk through the fundamentals of game theory and begin finding solutions for the prisoner's dilemma. We will talk more about strategy types and how to find them through the use of Nash Equilibrium. Afterwards we will explore optimal strategies for a variation on rock paper scissors and will end with a variation on prisoners dilemma with a focus on cooperative play.

Background

Games and Game Theory

In the most basic sense, a game is a relational model of actions and results taken by players for the purpose of incentive. Players are given a set of actions that can be taken, each with a resulting outcome. These actions are used by the players to further their progress towards a goal. In each game, a player can have a strategy, or set of moves, that will allow them to progress further than if they played normally. The fundamental principle of game theory is to discover these strategies. More formally, game theory is the mathematical modeling of the consequences of conflicts, cooperations, and actions between players. A player's set of actions can lead to a variety of different outcomes. For most basic games, the costs/benefits of the outcomes that arise can be modeled in a payoff matrix. This matrix, also known as Normal Form, indicates the utility or payoff that the player can gain from playing each possible action against an opponent.

One of the most prominent games studied by game theorists is the prisoner's dilemma. The rules of which are:

Two members of a criminal gang are arrested and imprisoned. Each prisoner is in solitary confinement with no means of communicating with the other. The prosecutors lack sufficient evidence to convict the pair on the principal charge. They hope to get both sentenced to a year in prison on a lesser charge. Simultaneously, the prosecutor's offer each prisoner a bargain. Each prisoner is given the opportunity either to: betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent. The maximum prison sentence of the crime is 5 years. The offer is:

- If A and B each betray the other, each of them get 1 year taken off their prison sentence
- If A betrays B but B remains silent, A will be set free and B will serve the full 5 years in prison (and vice versa)
- If A and B both remain silent, both of them will take off 3 years from their prison sentence (on the lesser charge)

[Modified from: https://en.wikipedia.org/wiki/Prisoner%27s_dilemma (https://en.wikipedia.org/wiki/Prisoner%27s_dilemma)]

Mathematical Model

Defining Terms and Notations

In the following parts of the report, we will be using some new notations to define our problems. In game theory, there are two main types of strategies that are analyzed: Pure and Mixed. Pure strategies are very straight-forward. In a **pure strategy**, a players selects a single strategy and plays it. Each row or column of a payoff matrix represents both an action and a pure strategy. In a **mixed strategy**, you randomize over the set of available actions and choose one. Before going further into developing strategies, we should define some notations.

- Let there be an n number of players, player-1, player-2, ... , player-n.
- A **strategy profile** is a set of n strategies, one strategy played by each player. This is defined as $S = (s_1, \dots, s_n)$.
- A **action profile** is a set of n actions, one action taken by each player. This is defined as $A = (a_1, \dots, a_n)$.
- **Utility** is the payoff of taking a certain action or strategy. This is defined by U.
- Let $p_i(a_j)$ = the probability that action a_j will be played in a mixed strategy i.
- **Expected utility** or the expected payoff of a mixed strategy for player i can be defined as a function U_i
  - If $S = (s_1, \dots, s_n)$ is a mixed strategy profile
  - For every action profile ($a_1, a_2, \dots, a_n$), multiply it's probability and utility.
$$U_i(a_1, a_2, \dots, a_n) * p_1(a_1) * p_2(a_2) * \dots * p_n(a_n)$$
  - The total expected utility for player i is:

$$U_i(s_1, \dots s_n) = \sum_{(a_1,\dots,a_n)\in A} U_i(a_1, \dots, a_n) * p_1(a_1) * \dots * p_n(a_n)$$

Dominant Strategy Equilibrium

In a general game, there are multiple possible strategies for a player. For a strategy to be good, it must maximize the player's utility, also known as the reward. We can represent utility as a function $U_i(s_i)$ which is a player i's utility from strategy s_i. We let S_i = {the set of all possible strategies for a player-i} and s_i refers to a strategy in S_i. A strategy s_i (contained in S_i) dominates if with it player-i always does better or equally as well than with another strategy s_i^* (so only if $U_i(s_i) \geq U_i(s_i^*)$). s_i is the dominant strategy if it is better than all $s_i^* \in S_i$.

Our prisoner's dilemma problem is formally classified as an imperfect-information non-zero-sum problem. Imperfect-information meaning that neither players will know what the other player will do until after both players have taken an action. It is non-zero-sum in contrast to a zero-sum game where players benefit only at an equal expense of another. The payoff matrix for this game is:

$C_1 \backslash C_2$	**Ally**	**Betray**
Ally	3 \ 3	0 \ 5
Betray	5 \ 0	1 \ 1

(C_1 are the actions for criminal 1, C_2 are the actions of criminal 2, and table values are the amount of prison years to be taken off)

Looking at the payoff matrix, you can intuitively see the dominant strategy would be to always betray you opponent. Looking at the matrix, you can never do worse by betraying. If your opponent betrays, you get 0 years off for allying and 1 year off for betraying, so you should betray. If your opponent allies, you get 3 years off for allying and 5 years off for betraying, so you should betray.

Pure Strategy

To define our utility function, it would be easiest to define the possible situations:

- Let **A** be Betrayal Reward (score for betraying when opponent allies): **5**
- Let **B** be Ally Reward (score for both allying): **3**
- Let **C** be Betrayal Penalty (score for allying when opponent betrays): **1**
- Let **D** be Ally Penalty (score for both betraying): **0**

Assuming a strayegy profile (x,y) to be 0 for Ally and 1 for Betray, the utility function is defined as:

$$U = C * x * y + A * x * (1 - y) + D * y * (1 - x) + B * (1 - x) * (1 - y))$$

A players dominant strategy for an opponent playing 0 or 1 would be to maximize the utility. This can be represented in a linear programming (LP) problem.

Variables:

- For decisions in $[\ \ 0\ \ 1\ \ ]$, a decision variable x to indicate the player's strategy (0 for "Ally" and 1 for "Betray").
- A decision y from the opponent (0 for "Ally" and 1 for "Betray").

Constraints:

- Both x and y must be either 0 or 1.

Objective:

- Maximize the utility function U.

Model

$$\begin{aligned} \max_{x} \quad & xy + 5x(1 - y) + 3(1 - x)(1 - y)) \\ s.t. \quad & y \in [\ \ 0\ \ 1\ \ ] \\ & x \in [\ \ 0\ \ 1\ \ ] \end{aligned}$$

In [1]:
```
using JuMP, Mosek, Clp, NamedArrays
#=
situations for pure strategy
situation 1: opponent allies
situation 2: opponent betrays
=#
y = [0 1]

A = 5
B = 3
C = 1
D = 0;
```

In [3]:
```
# solving for a prisoners dilemma pure strategy
m = Model(solver = MosekSolver())

# binary variable for player action for each situation, 0 for ally, 1
for betray
@variable(m, x[1:2], Bin)
# variables for utility function
@variable(m, util[1:2])
# utility function for every situation
for i = 1:2
    @constraint(m, util[i] == C*x[i]*y[i] + A*x[i]*(1-y[i]) + D*y[i]*(
1-x[i]) + B*(1-x[i])*(1-y[i]))
end
# maximize the utility for all situations
@objective(m, Max, sum(util))

solve(m)
solution = getvalue(x)

println("Solutions: x = ", solution)
print("If the opponent chooses Ally, you should ")
if solution[1] == 0
    println("Ally")
else
    println("Betray")
end
print("If the opponent chooses Betray, you should ")
if solution[2] == 0
    println("Ally")
else
    println("Betray")
end
```

```
Solutions: x = [1.0,1.0]
If the opponent chooses Ally, you should Betray
If the opponent chooses Betray, you should Betray
```

Looking at the results, we can see that the optimal solution, regardless of what your opponent plays is to always betray. This can be seen intuitively, despite the globally superior option of both participants cooperating. Looking at the payoff matrix, no matter what the opponent plays, you will always do better than them by betraying.

While in the prisoner's dillema, since the optimal strategy is to always betray, a mixed strategy is not necessary; it does not matter what your opponent picks.

In what scenario, then, would a mixed strategy be viable?

Nash Equilibrium

Nash Equilibrium is a solution concept of a non-cooperative game involving two or more players in which each player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by changing only his or her own strategy. In other words, a group of players are in Nash equilibrium if each one is making the best decision possible, taking into account the decisions of the others in the game as long as the other party's decision remains unchanged. We can model the Nash Equilibrium through a linear programming maximin problem. This is a problem where you want to maximize the lower bound of the expected utility you can gain.

Minimax and Maximin

The Nash Equilibrium formally defined as a Linear Programming Minimax/Maximin problem. In a minimax problem, you want to minimize your losses. In a game where a benefit to the player results in a loss for the opponent, this means setting up a upper bound on the utility your opponent gets, allowing you to lower the value they are sure to get, thereby limiting their potential for points. In a maximin problem, you want to maximize the lower bound on your score, allowing you to increase the value you are sure to get, thereby increasing your potential for points. To model this, we should first define some terms.
Define:

- Let u be a strategy set for player-1, $[u_1 u_2 \dots u_n]$
- Let every strategy $0 \leq u_i \leq 1$ for $u_i \in u$
- and $\sum_{(u_1,\dots,u_n)\in u} u_i = 1$
- The same goes for player-2's strategy set v.
- We define player-1's payoff in a payoff matrix P.
- Player-1's utility is then defined by a linear combination of the payoff matrix, and the two player's strategies: uPv^T.
- Player-1 will try to maximize a lower bound μ on their minimum expected utility.
- For a zero-sum game, player-2 optimal strategy would be to minimize the upper bound for player-1's expected utility. The prisoner's dilemma is not a zero-sum game, however since a benefit to player-1 results in a deficit for player-2 and vice versa, this method is still optimal.

Player-1's Nash Equilibrium will be modeled by:

Variables:

- A decision variable u where u_i indicates the player's probability of taking strategy i.
- A threshold μ to limit the lower bounds.

Constraints:

- No $u_i \in u$ can be negative.
- The sum of all $u_i \in u$ must equal 1.

- The threshold μ must be less than the expected utility.

Objective:

- Maximize the threshold μ.

Model:

$$\begin{aligned} &\max_{u,\mu} \mu \\ s.t. \quad &u \geq 0 \\ &\sum u = 1 \\ &P^T u \geq \mu \end{aligned}$$

and player-2's Nash Equilibrium will be modeled by:

Variables:

- A decision variable v where v_i indicates the player's probability of taking strategy i.
- A threshold t to limit the upper bounds.

Constraints:

- No $v_i \in v$ can be negative.
- The sum of all $v_i \in v$ must equal 1.
- The threshold t must be greater than the expected utility.

Objective:

- Minimize the threshold t.

Model:

$$\begin{aligned} &\max_{v,t} t \\ s.t. \quad &v \geq 0 \\ &\sum v = 1 \\ &Pv \leq t \end{aligned}$$

Let's revisit the previous problem with this solution context in mind. To find the game state when both players use their optimal strategy, we have player-1 selecting a row and player-2 selecting a column. The rows and columns are 0 for choosing "Ally" and 1 for choosing "Betray".

In [4]:
```
# payoff matrix for player 1
P = [ 3 0;
      5 1 ]

# solving for a nash equilibrium, player-1 wants to maximize their pay
out
m_u = Model(solver = ClpSolver())
```

```
# variable for player 1's actions
@variable(m_u, u[1:2] >= 0)

# maximin variable
@variable(m_u, μ)

# choose one row (Ally or Betray)
@constraint(m_u, sum(u) == 1)

# set a threshold value
@constraint(m_u, P'*u .>= μ)

# maximize the threshold
@objective(m_u, Max, μ)

# get optimal choice
solve(m_u)
uopt = getvalue(u)

#########################################################################
########

# solving for a nash equilibrium, player-2 wants to minimize player-1'
s payout
m_v = Model(solver = ClpSolver())

# binary variable for player 1's actions
@variable(m_v, v[1:2] >= 0)

# minimax variable
@variable(m_v, t)

# choose 1 column (Ally or Betray)
@constraint(m_v, sum(v) == 1)

# set a threshold value
@constraint(m_v, P*v .<= t)

# minimize the threshold
@objective(m_v, Min, t)

# get optimal choice
solve(m_v)
vopt = getvalue(v)

# display nash equilibrium
cellopt = uopt*vopt'
out = NamedArray(cellopt,(["Ally", "Betray"],["Ally", "Betray"]),("Pla
yer 1", "Player 2"))
```

```
Out[4]: 2×2 Named Array{Float64,2}
        Player 1 ╲ Player 2 │   Ally  Betray
        ────────────────────┼──────────────
        Ally                │    0.0     0.0
        Betray              │    0.0     1.0
```

As predicted, the Nash Equilibrium Strategy is for both players to Betray.

If expressed probablistically, the best strategy of the Prisoner's dilemma can then be characterized as a single Nash Equilibrium, with both players having a (0%,100%) strategy of Ally or Betray, respectively.

Consider the situation where no utility is gained by betraying your partner. This is a variation of the Prisoner's Dilemma called the "Stag Hunt" scenario. Two hunters are starving and must choose to hunt Stags or Rabbits. Each player can kill a Rabbit, worth 1 unit of meat, but must cooperate to down a Stag, worth 4 units. All units are split between the hunters.

$H_1 \backslash H_2$	**Stag**	**Rabbit**
Stag	2 \ 2	0.5 \ 0.5
Rabbit	0.5 \ 0.5	1 \ 1

(H_1 are the actions for Hunter 1, H_2 are the actions of Hunter 2, and table values are the amount of meat collected after the hunt)

In this scenario there is not one, but three Nash Equilibria: (0%,100%), (100%,0%), and a third that resolves in an indifference towards either. The third is accomplished via a mixed strategy.

Mixed Strategy

Again, let us define possible situations for the utility function:

- Let **A** be Stag Reward (both players hunt Stag): **2**
- Let **B** be Rabbit Reward (both players hunt Rabbit): **1**
- Let **C** be Mismatch Penalty (players choose different options): **0.5**

Assuming a set of decisions (x,y) to be probability of hunting stag (0.5 means 50% probability, 0.25 means 25%) the payoff matrix is defined as:

$$P = \begin{bmatrix} 2 * p(s) & 0.5 * (1 - p(s)) \\ 0.5 * p(s) & 1 * (1 - p(s)) \end{bmatrix}$$

The player's optimal strategy is found through the Nash Equilibrium.

In [7]:
```
# probabilities of opponent choosing to hunt stag
y_p = [ 0.1 0.25 0.375 0.50 0.75 ];

# payoff matrix for player 1
P = [ 2 0.5;
      0.5 1 ]

# payoff matricies when opponents probabilities are y_p
X = zeros(2,2,5)
for i = 1:5
    X[:,:,i] = hcat(P[:,1]*y_p[i],P[:,2]*(1-y_p[i]))
end

# solving for a nash equilibrium, player-1 wants to maximize their pay
out
m_p = Model(solver = ClpSolver())

# variable for player 1's actions
@variable(m_p, u[1:5,1:2] >= 0)

# maximin variable
@variable(m_p, μ[1:5])

for i = 1:5
    # choose one row (Ally or Betray)
    @constraint(m_p, sum(u[i,:]) == 1)
end

for i = 1:5
    # set a threshold value
    @constraint(m_p, X[:,:,i]'*u[i,:] .>= μ[i])
end
# maximize the threshold
@objective(m_p, Max, sum(μ))

# get optimal choice
solve(m_p)
uopt = getvalue(u)

for i = 1:5
    println("Partner has a: ")
    println("-> ", round(y_p[i]*100,1), "% chance to hunt rabbit")
    println("-> ", round((1-y_p[i])*100,1), "% chance to hunt stag")
    println("You should hunt stag: ", round(uopt[i,1]*100,1), "% of th
e time")
    println("and hunt rabbit: ", round((1-uopt[i,1])*100,1), "% of the
time")
    println()
end
```

```
Partner has a:
-> 10.0% chance to hunt rabbit
-> 90.0% chance to hunt stag
You should hunt stag: 100.0% of the time
and hunt rabbit: 0.0% of the time

Partner has a:
-> 25.0% chance to hunt rabbit
-> 75.0% chance to hunt stag
You should hunt stag: 83.3% of the time
and hunt rabbit: 16.7% of the time

Partner has a:
-> 37.5% chance to hunt rabbit
-> 62.5% chance to hunt stag
You should hunt stag: 50.0% of the time
and hunt rabbit: 50.0% of the time

Partner has a:
-> 50.0% chance to hunt rabbit
-> 50.0% chance to hunt stag
You should hunt stag: 25.0% of the time
and hunt rabbit: 75.0% of the time

Partner has a:
-> 75.0% chance to hunt rabbit
-> 25.0% chance to hunt stag
You should hunt stag: 0.0% of the time
and hunt rabbit: 100.0% of the time
```

Zero-Sum Games

As mentioned before, the two-player coordination model is a non-zero-sum game, which describes a situation in which the interacting parties' aggregate gains and losses can be less than or more than zero. In constrast a zero-sum game is, in game theory and economics, a mathematical representation of a situation where each participant's gain or loss of utility is exactly balanced by the losses or gains of the other participants.

Zero-sum games are most often solved with the minimax theorem which is closely related to linear programming duality, or with Nash equilibrium.

Classic RPS

A great example of a classic zero-sum game is rock-paper-scissors. Players choose between the three and the winner gets a point and the loser loses a point. Rock beats Scissors, Scissors beat Paper, and Paper beats Rock. It should be noted, as shown below, that the utility of each option always comes at the expense of the other player, such that the net utility will always add up to 0.

$P_1 \backslash P_2$	**Rock**	**Paper**	**Scissors**
Rock	0 \ 0	-1 \ 1	1 \ -1
Paper	1 \ -1	0 \ 0	-1 \ 1
Scissors	-1 \ 1	1 \ -1	0 \ 0

It should go without saying that, assuming the opponent has an even disposition towards all options (mixed strategy), RPS is not a practical problem to solve. All options have equal utility, and a globally best option cannot be determined. This means each player will win 50% of the time, resulting in a net utility of 0 for each player. We can work around this by altering the ruleset.

Betrayal Rock Paper Scissors

Similar rules apply as in normal rock paper scissors. When you win, you get 1 point and when you lose, you lose 1 point. However, in the “Betrayal” version, you can declare what you will play beforehand and your opponent does the same. If you declare "Rock", then you tell the truth when you play rock and lie when you play something else. If you win “Rock, Paper, Scissors” when you tell the truth (i.e. declare “Rock” and play “Rock”), then you win 2 bonus points (for a total of 3 points). If you tell the truth but lose, there is no additional penalty (so you just lose 1 point). If you lie (i.e. declare “Rock” but play “Scissors”) and win, you don’t get any bonus points (so you gain just 1 point). If you and your opponent tell the truth and you lose (i.e. they declare “Paper” and play “Paper” while you declare “Rock” and play “Rock”), then you lose 1 additional point (so you lose 2 points total) . If you lie, your opponent tells the truth and your opponent wins (i.e. they declare “Paper” and play “Paper” while you declare “Paper” and play “Rock”), you lose 1 additional point for lying and 1 additional point for losing while the opponent telling the truth (for a total loss of 3 points). Assuming you declare “Rock” and your opponent declares “Paper”, the payoff matrix would look like:

Betrayal RPS: P_1 bets on Rock, P_2 bets on Paper

$P_1 \backslash P_2$	**Rock**	**Paper**	**Scissors**
Rock	0 \ 0	-2 \ 5	5 \ -3
Paper	1 \ -2	0 \ 0	-2 \ 1
Scissors	-2 \ 1	1 \ -1	0 \ 0

In [8]:
```
action = [ "rock" "paper" "scissors" ]
m = Model(solver = MosekSolver())
```

```
# Payoff for originally rock paper scissors
P_o = [ 0 -1 1;
        1 0 -1;
        -1 1 0]

# probabilities of opponent choosing [ r p s ]
p = [ 1/3 1/3 1/3 ]

# Payoff for declaring rock
# Opponent declares rock
P_r_r = [ 0 -1 5;
          1 0 -2;
         -3 1 0 ]
# Opponent declares paper
P_r_s = [ 0 -1 5;
          1 0 -3;
          -2 1 0 ]
# Opponent declares scissors
P_r_p = [ 0 -2 5;
          1 0 -2;
          -2 1 0 ]
# Opponent declares rock paper or scissors with probabilities p
P_r = p[1]*P_r_r + p[2]*P_r_p + p[3]*P_r_s

# Payoff for declaring paper
# Opponent declares rock
P_p_r = [ 0 -2 1;
          5 0 -1;
          -3 1 0]
# Opponent declares paper
P_p_p = [ 0 -3 1;
          5 0 -1;
          -2 1 0]
# Opponent declares scissors
P_p_s = [ 0 -2 1;
          5 0 -2;
          -2 1 0]
# Opponent declares rock paper or scissors with probabilities p
P_p = p[1]*P_p_r + p[2]*P_p_p + p[3]*P_p_s

# Payoff for declaring scissors
# Opponent declares rock
P_s_r = [ 0 -2 1;
          1 0 -2;
          -2 5 0 ]
# Opponent declares paper
P_s_p = [ 0 -3 1;
          1 0 -2;
          -1 5 0]
# Opponent declares scissors
```

```
P_s_s = [ 0 -2 1;
          1 0 -3;
          -1 5 0]
# Opponent declares rock paper or scissors with probabilities p
P_s = p[1]*P_s_r + p[2]*P_s_p + p[3]*P_s_s

# P_A holds the payoff matricies for declaring rock paper and scissors
P_A = zeros(3,3,3)
P_A[:,:,1] = P_r
P_A[:,:,2] = P_p
P_A[:,:,3] = P_s

##########################################################################
###

# solving for a nash equilibrium, player-1 wants to maximize their pay
out
m_rps = Model(solver = ClpSolver())

# variable for player 1's actions
@variable(m_rps, rps[1:3,1:3] >= 0)

# maximin variable
@variable(m_rps, μ[1:3])

for i = 1:3
    # choose (rock paper or scissors) at a probability
    @constraint(m_rps, sum(rps[i,:]) == 1)
end

for i = 1:3
    # set a threshold value
    @constraint(m_rps, P_A[:,:,i]'*rps[i,:] .>= μ[i])
end

# maximize the threshold
@objective(m_rps, Max, sum(μ))

# get optimal choices
solve(m_rps)
rpsopt = getvalue(rps)

println("Opponent has a: ")
for i = 1:3
    println("-> ", round(p[i]*100,1), "% chance to declare ", action[i
])
end
println()
for j = 1:3
    println("If you declare ", action[j], " you should play: ")
```

```
    for k = 1:3
        println("-> ", action[k], " ", round(rpsopt[j,k]*100,1), "% of
the time.")
    end
    println()
end
```

```
Opponent has a:
-> 33.3% chance to declare rock
-> 33.3% chance to declare paper
-> 33.3% chance to declare scissors

If you declare rock you should play:
-> rock 23.1% of the time.
-> paper 52.0% of the time.
-> scissors 24.9% of the time.

If you declare paper you should play:
-> rock 24.9% of the time.
-> paper 23.1% of the time.
-> scissors 52.0% of the time.

If you declare scissors you should play:
-> rock 52.0% of the time.
-> paper 24.9% of the time.
-> scissors 23.1% of the time.
```

We can apply this model to sets of different probabilities where we can see the effects of how the opponent's decisions can affect ours:

When:
p = [3/4 1/4 1/4]

Opponent has a:
-> 75.0% chance to declare rock
-> 25.0% chance to declare paper
-> 25.0% chance to declare scissors

If you declare rock you should play:
-> rock 23.0% of the time.
-> paper 54.0% of the time.
-> scissors 23.0% of the time.
Utility threshold at: -0.0599279835390946

If you declare paper you should play:
-> rock 25.0% of the time.
-> paper 25.0% of the time.
-> scissors 50.0% of the time.
Utility threshold at: -0.06250000000000001

If you declare scissors you should play:
-> rock 52.0% of the time.
-> paper 27.0% of the time.
-> scissors 21.0% of the time.
Utility threshold at: -0.09102972399150744

To maximize utility, you should follow the strategy of declaring rock.

When:
p = [1/8 5/8 2/8]

Opponent has a:
-> 12.0% chance to declare rock
-> 62.0% chance to declare paper
-> 25.0% chance to declare scissors

If you declare rock you should play:
-> rock 21.0% of the time.
-> paper 51.0% of the time.
-> scissors 27.0% of the time.
Utility threshold at: -0.07407020476389456

If you declare paper you should play:
-> rock 23.0% of the time.
-> paper 22.0% of the time.
-> scissors 55.0% of the time.
Utility threshold at: -0.050560672807368846

If you declare scissors you should play:
-> rock 50.0% of the time.
-> paper 24.0% of the time.
-> scissors 25.0% of the time.
Utility threshold at: -0.043982448809026256

To maximize utility, you should follow the strategy of declaring scissors.

As we can see, simply declaring intention at the beginning has a drastic effect on your best line of play, and more importantly, turns RPS into a solvable scenario.

Now, let us bring this full-circle.

The Zero Escape Dilemma

The Zero Escape Problem is roughly based on an expanded Prisoner's Dilemma. The game is as follows:

You and 4 other people (5 total) are locked inside a prison. Inside, there is a single button labeled "Escape". The game is played in rounds, where the prisoners are each paired off with each other and play the coordination game (Ally/Betray). Thus, one round consists of 10 games, with each prisoner meeting each other prisoner once.

As with the normal prisoner's dilemma:

- If A and B each 'Betray' the other, each receives 0 points
- If A and B 'Ally' with each other, each receives 2 points
- If A 'Betray's B, but B 'Ally's, A gains 3 points and B loses 2 points, and vice versa

The additional conditions are:

- All players begin with 3 points
- Once a player reaches 9 points, they may press the "Escape" button and all players with 9 or more points may leave

And there is a catch:

- As soon as a player leaves, all players with less than 9 points immediately die
- There is one "evil" player that everyone is aware of, with a 50% chance to betray. They will press the button immediately upon reaching 9 points
- Players cannot identify each other when playing the coordination game

Point totals are calculated in parallel at the end of the round. It is assumed that all but the "evil" player are cooperative to the group's survival. Thus, your goal in this problem is to have all cooperative prisoners escape. If no prisoner has won by the time a round has finished, a new round will start. The utility matrix for an individual game is shown below:

$P_1 \backslash P_2$	**Ally**	**Betray**
Ally	2 \ 2	-2 \ 3
Betray	3 \ -2	0 \ 0

We already know that the global optimal strategy of the prisoner's dilemma is to betray 100% of the time, and this holds true in this scenario as well if one looks strictly at the utility matrix. What separates this game from the prisoner's dilemma is that a player, similar to the stag hunt, is forced into global cooperation.

So, what is the optimal strategy for this game? If all players choose to Ally 100% percent of the time, this means that the "evil" player will only need a single round to reach 9, and thus eliminate the group as each can only have a maximum of 7 points.

In [9]:
```
z = [ 2 3 4 5 6 ]
# First, let us speculate on an individual's chance of escape
m = Model(solver = MosekSolver())

# The problem can viewed from a probablistic standpoint, as in the zer
o-sum games
# Each player is guaranteed to meet with evil at least once out of fou
r games.
# Thus, each player needs to maximize the chance of "Betray"ing the ba
d guy, while maintaining
# a net positive increase among themselves.

# x represents number of times player should betray per round
@variable(m, 4 >= x[1:5] >= 1)
# Maintain a net positive score in worst-case scenario

@expression(m, netPositive[i = 1:5],  (4-x[i])*2-(x[i]-1))

for i = 1:5
    @constraint(m, netPositive[i] >= z[i])
end
# minimize the chance of being betrayed while allying
@objective(m, Min, sum((4-x)/4*0.125))

solve(m)
for i = 1:5
    println("To maintain a net positive score of ", z[i])
    print("Betray: ",round(getvalue(x[i]),3)," times out of four or ")
    println(round(100*getvalue(x[i])/4,2),"%\n")
end
```

```
To maintain a net positive score of 2
Betray: 2.333 times out of four or 58.33%

To maintain a net positive score of 3
Betray: 2.0 times out of four or 50.0%

To maintain a net positive score of 4
Betray: 1.667 times out of four or 41.67%

To maintain a net positive score of 5
Betray: 1.333 times out of four or 33.33%

To maintain a net positive score of 6
Betray: 1.0 times out of four or 25.0%
```

Regardless of score maintained, it is clear that the utility matrix does not accurately represent the value each decision actually has for the cooperative players. If that were the case, we would not be seeing variance in the number of times a player should betray per turn. Rather, the optimal strategy would be to always betray. We can work backwards and use the probability found above to determine the true, scaled utility.

In [10]:
```
# Mixed strategy probabilities to Betray
y_z = [ 0.5833 0.5 0.4167 0.3333 0.25 ];

Payoff = [ 2 -2;
           3 0 ]

X_z = zeros(2,2,5)
for i = 1:5
    X_z[:,:,i] = hcat(Payoff[:,1]*y_z[i],Payoff[:,2]*(1-y_z[i]))
end

for i = 1:5
    println("Player-1's payoff matrix at ", round(y_z[i]*100,2), "% be
trayal probability is: ")
    println(" ", round(X_z[1,1,i],2), "   ", round(X_z[1,2,i],2))
    println(" ", round(X_z[2,1,i],2), "   ", round(X_z[2,2,i],2))
end
```

```
Player-1's payoff matrix at 58.33% betrayal probability is:
 1.17   -0.83
 1.75   0.0
Player-1's payoff matrix at 50.0% betrayal probability is:
 1.0   -1.0
 1.5   0.0
Player-1's payoff matrix at 41.67% betrayal probability is:
 0.83   -1.17
 1.25   0.0
Player-1's payoff matrix at 33.33% betrayal probability is:
 0.67   -1.33
 1.0   0.0
Player-1's payoff matrix at 25.0% betrayal probability is:
 0.5   -1.5
 0.75   0.0
```

Conclusion

On a more profound note, the difference between these two games is betrayal versus altruism. Could the implication be that being good doesn't pay?

More seriously, game theory is a very well defined subset of economics, but its applications aren't necessarily stale. In the end, the goal is to break down a challenge in order to construct a better one, and that is exactly what was done here.

Using our findings from applying Nash Equilibrium to a series of games and situations. As we can see from optimization through Nash Equilibrium, by using minimax theorem, we can optimize two player's strategies against eachother and find an equilibrium point for the game. For our prisoner's dilemma, we find the equilibrium to be having both players betray. In a cooperative play however, things change.

By assigning a "Stag Hunt" ultimatum to an iterated prisoner's dilemma we can observe a difference in collaborative utility, outside of the standard decision matrix. One might argue than in a populous setting, a tit-for-tat strategy would be more appropriate than a pure strategy, however we find that is not the case. Strategy is a function of utility, but utility might not always be a function of strict gains.

Works Cited

Nash Equilibrium

http://www.dis.uniroma1.it/~facchinei/didattica/giochi/Lettura_3.pdf (http://www.dis.uniroma1.it/~facchinei/didattica/giochi/Lettura_3.pdf)

General Game Theory

https://www.cs.cmu.edu/~ggordon/780-fall07/fall06/lectures/Game_theory_I.pdf (https://www.cs.cmu.edu/~ggordon/780-fall07/fall06/lectures/Game_theory_I.pdf)

Game Theory, Decision Thresholds

http://www.laurentlessard.com/bookproofs/tag/game-theory/ (http://www.laurentlessard.com/bookproofs/tag/game-theory/)

RPS Optimization

https://optimization.mccormick.northwestern.edu/index.php/Matrix_game_(LP_for_game_theory (https://optimization.mccormick.northwestern.edu/index.php/Matrix_game_(LP_for_game_theory))

https://www.cs.ubc.ca/~kevinlb/teaching/cs532l%20-%202011-12/lectures/lect5a.pdf (https://www.cs.ubc.ca/~kevinlb/teaching/cs532l%20-%202011-12/lectures/lect5a.pdf) http://www.mathematik.uni-wuerzburg.de/~heusinger/DRegNIP.pdf (http://www.mathematik.uni-wuerzburg.de/~heusinger/DRegNIP.pdf) http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6819427 (http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6819427)